View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

Characterizing functional dependencies in formal concept analysis with pattern structures

Jaume Baixeries, Mehdi Kaytoue, Amedeo Napoli

To cite this version:

Jaume Baixeries, Mehdi Kaytoue, Amedeo Napoli. Characterizing functional dependencies in formal concept analysis with pattern structures. Annals of Mathematics and Artificial Intelligence, Springer Verlag, 2014, 72, pp.129 - 149. 10.1007/s10472-014-9400-3 . hal-01101107

HAL Id: hal-01101107

https://hal.inria.fr/hal-01101107

Submitted on 9 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Annals of Mathematics and Artificial Intelligence manuscript No.
(will be inserted by the editor)

Characterizing Functional Dependencies in Formal Concept Analysis with Pattern Structures

Jaume Baixeries · Mehdi Kaytoue · Amedeo Napoli

Received: date / Accepted: date

Abstract Computing functional dependencies from a relation is an important database topic, with many applications in database management, reverse engineering and query optimization. Whereas it has been deeply investigated in those fields, strong links exist with the mathematical framework of Formal Concept Analysis. Considering the discovery of functional dependencies, it is indeed known that a relation can be expressed as the binary relation of a formal context, whose implications are equivalent to those dependencies. However, this leads to a new data representation that is quadratic in the number of objects w.r.t. the original data. Here, we present an alternative avoiding such a data representation and show how to characterize functional dependencies using the formalism of pattern structures, an extension of classical FCA to handle complex data. We also show how another class of dependencies can be characterized with that framework, namely, degenerated multivalued dependencies. Finally, we discuss and compare the performances of our new approach in a series of experiments on classical benchmark datasets.

Keywords: Association rules, attribute implications, data dependencies, pattern structures, formal concept analysis

Jaume Baixeries
Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya. 08032 Barcelona. Catalonia.
E-mail: jbaixer@lsi.upc.edu

Mehdi Kaytoue
Université de Lyon. CNRS, INSA-Lyon, LIRIS. UMR5205, F-69621, France.
E-mail: mehdi.kaytoue@insa-lyon.fr

Amedeo Napoli
LORIA (CNRS – Inria Nancy Grand-Est – Université de Lorraine)
B.P. 239, Équipe Orpailleur, Bâtiment B, F-54506 Vandœuvre-lès-Nancy, France.
E-mail: amedeo.napoli@loria.fr

1 Introduction

The discovery of functional dependencies is an important topic in the database field since they represent the fact that the value of one or several attributes is uniquely (functionally) determined by the values of other attributes. As such, they are valuable in order to explain the normalization of a database schema in the Relational Database Model. For example, consider the relation $AddressBook(id, name, street, ZIP, City)$: it entails the functional dependencies stating that any two tuples of this relation that have the same value of ZIP code, also have the same value for the attribute $City$. Formally, given a relation schema $\mathcal{U}$, i.e. a set of attributes to describe some objects or tuples, a functional dependency is denoted by $X \to Y$, $X, Y \subseteq \mathcal{U}$ and means that the objects that take the same values for the attributes in X take also the same values for the attributes in Y. Table 1 in Section 2 is a tabular representation of a relation. Rows denote objects (or tuples) and columns denote attributes of the schema. There, the functional dependency $a \to d$ holds: when t_1 and t_3 take the same value for the attribute a, they also take a same value for the attribute d. In the relational database model there are different types of dependencies (conditional [15], impurity [34], DMVDs [33], etc., see [21] for a more detailed survey), although functional dependencies are among the most popular, and have been widely studied [1,35,28,32,39].

Besides, functional dependencies, and dependencies in general, are closely linked to attribute implications in Formal Concept Analysis [17]. FCA is an important mathematical framework rooted in lattice theory that is also used for data-analysis purposes (deeply described in [17]). Among other, it aims at discovering implicit relations between objects and their attributes. It starts with a triple (G, M, I), called a formal context, where G is a set of objects, M a set of attributes and I a binary relation such as $I \subseteq G \times M$. The tabular representation of a binary relation is given in Figure 1 in Section 2, where rows denote objects, columns denote attributes, and a cross denotes an element of the relation. So-called implications are expressions of the form $X \to Y$, $X, Y \subseteq M$ stating that when an object has attributes in X, then it has also attributes in Y. In the formal context of Figure 1, the implication $m_1 \to m_2$ is the only one that holds.

As such, functional dependencies (FDs) and attribute implications are expressions of the same form, i.e. $X \to Y$, defined over a set of attributes. However, in the first case, FDs are defined on numerical or categorical attributes, while implications are defined on binary attributes. Thus, to show an equivalence (or just links) between FDs and implications, the original data in which FDs hold have to be transformed into a formal context, whose implications can then be compared to FDs. This was actually presented in the book of FCA (see [17], page 92) and as well in [27]. It was shown how to build a formal context from the original data and that the implications in this formal context are syntactically equivalent to the FDs of the original data. The second table in Figure 2 shows the formal context obtained from the original data in Table 1: whereas the procedure is explained later, one should notice that indeed

the implication $a \rightarrow d$ holds, which is also a FD in the original data. Unfortunately, the number of objects of the resulting context is quadratic w.r.t. the original, which does not allow this method to be applied on large datasets.

The previous remark is actually the motivation of the present work leading to the following question: *Can we characterize with FCA functional dependencies as implications, avoiding a significantly larger data representation?* We positively answer this question by introducing a method based on Pattern Structures [16]. A pattern structure can be understood as a generalization of standard FCA to handle complex data (say, non binary): instead of a binary relation between some objects and their attributes, it applies on a relation between objects and their descriptions that form a particular partially ordered set. Our approach consists in considering that the attributes from the original relation schema $\mathcal{U}$ can be described by a partition over the set of tuples, and that the set of partitions forms a lattice. As such, so-called *partition pattern structures* are introduced in this paper, and we show that the implications they hold are equivalent to the functional dependencies, as well as the attribute implications holding in the formal context introduced in the previous section.

Consequently, our contribution is three-fold:

- Firstly, we present a new conceptual structure, called partition pattern structure.
- Secondly, we show how such a structure can be built from a numerical dataset to characterize functional dependencies: The interest is to prove that pattern structures are a flexible mechanism within FCA to encode the semantics of the dependencies without a heavy data representation.
- Finally, we show that this method allows one, with a minor variation, to characterize another kind of dependencies called degenerated multi-valued dependencies (DMVDs, introduced later). We also propose experiments showing that our conceptual structure has better computational properties than the classical FCA approach.

The paper is organized as follows. Basics on functional dependencies are presented in Section 2. Formal Concept Analysis and its usage to characterize functional dependencies is described in Section 3. We discuss data transformation with FCA in Section 3.3. It is followed by our main contribution which consists in characterizing functional dependencies with pattern structures (Section 4). Section 5 handles the case of degenerated multivalued dependencies. Before concluding, we compare our new approach in a series of experiments on classical benchmark datasets in Section 6.

2 Functional and Degenerated Multivalued Dependencies

We first introduce functional dependencies (FDs). Let $\mathcal{U}$ be a set of attributes, and let Dom be a set of values (a domain). For sake of simplicity, we assume that Dom is a numerical set. A tuple t is a function $t : \mathcal{U} \mapsto Dom$, and a table

T is a set of tuples. Usually tables are presented as a matrix, as in Table 1, where the set of tuples (or objects) is $T = \{ t_1, t_2, t_3, t_4 \}$ and $\mathcal{U} = \{ a, b, c, d \}$ is the set of attributes. We use *table, dataset, set of tuples* as equivalent terms. We overload the functional notation of a tuple in such a way that, given a tuple $t \in T$, we say that $t(X)$ (for all $X \subseteq \mathcal{U}$) is a tuple with the values of t in the attributes $x_i \in X$:

$$t(X) = \langle t(x_1), t(x_2), \ldots, t(x_n) \rangle$$

For example, we have that $t_2(\{ a, c \}) = \langle t_2(a), t_2(c) \rangle = \langle 4, 4 \rangle$. In the paper, the set notation is dropped: instead of $\{ a, b \}$ we use ab.

id	a	b	c	d
t_1	1	3	4	1
t_2	4	3	4	3
t_3	1	8	4	1
t_4	4	3	7	3

Table 1 An example of a table T, i.e. a set of tuples

Definition 1 ([39]) Let T be a set of tuples, and $X, Y \subseteq \mathcal{U}$. A **functional dependency (FD)** $X \to Y$ holds in T if:

$$\forall t, t' \in T : t(X) = t'(X) \implies t(Y) = t'(Y)$$

For instance, the functional dependencies $a \to d$ and $d \to a$ hold in T, whereas the functional dependency $a \to c$ does not hold since $t_2(a) = t_4(a)$ but $t_2(c) \neq t_4(c)$.

We now present a generalization of functional dependencies: degenerated multivalued dependencies.

Definition 2 ([33]) Let $X, Y, Z \subseteq \mathcal{U}$ of a table T, such that $X \cap Y = X \cap Z = Y \cap Z = \emptyset$ and $X \cup Y \cup Z = \mathcal{U}$. We say that a **degenerated multivalued dependency (DMVD)** $X \to Y$ holds in T if and only if:

$$\forall t, t' \in T : t(X) = t'(X) \implies t(Y) = t'(Y) \text{ or } t(\mathcal{U} \setminus X \setminus Y) = t'(\mathcal{U} \setminus X \setminus Y)$$

For instance, we have that $a \to b$ holds in the example table T, since $t_1(a) = t_3(a)$ and $t_1(cd) = t_3(cd)$, and $t_2(a) = t_4(a)$ and $t_2(b) = t_4(b)$. We remark that the functional dependency $a \to b$ does not hold in T, because of the pair of tuples t_1, t_3. Degenerated multivalued dependencies are a generalization of functional dependencies: if we drop the clause $t(\mathcal{U} \setminus X \setminus Y) = t'(\mathcal{U} \setminus X \setminus Y)$, we have the definition of functional dependencies. Therefore, if the functional dependency $X \to Y$ holds, then, the degenerated multivalued dependencies $X \to Y$ and $X \to \mathcal{U} \setminus X \setminus Y$ hold as well, whereas the opposite is not necessarily true, as the previous example shows.

Dependencies have a set of axioms stating which dependencies hold given an arbitrary set of dependencies of the same kind. The set of dependencies Σ closed under their own set of axioms is denoted by Σ^+. A minimal set of dependencies from which all other dependencies can be deduced by means of those axioms is called a *minimal generating set*.

Let $\mathcal{U}$ be the set of attributes of a relational table. The axioms for functional dependencies follow Armstrong rules [39] for all $X, Y, Z \subseteq \mathcal{U}$:

$$\frac{Y \subseteq X}{X \to Y} \qquad \frac{X \to Y}{X \cup Z \to Y} \qquad \frac{X \to Y, Y \to Z}{X \to Z}$$

These axioms are respectively called reflexivity, augmentation and transitivity. Implications also share the same axioms [17]. On the other hand, the axioms for degenerated multivalued dependencies (DMVDs) are reflexivity, complementation or symmetry, augmentation and transitivity, i.e. for all $X, Y, Z, V, W \subseteq \mathcal{U}$:

$$\frac{Y \subseteq X}{X \to Y} \qquad \frac{X \to Y}{X \to \mathcal{U} \setminus Y \setminus X} \qquad \frac{X \to Y, V \subseteq W}{W \cup X \to V \cup Y} \qquad \frac{X \to Y, Y \to Z}{X \to Z \setminus Y}$$

These axioms are also shared by multivalued dependencies, a well-known kind of dependencies in the relational database model [33].

In this paper, we will use pattern structures to characterize a set of functional dependencies (and DMVDs) that hold in data table. In fact, it is not an arbitrary Σ, but precisely Σ^+. Firstly, we recall how such characterization is classically done in the FCA literature.

3 The characterization of Functional Dependencies within FCA

In this section, we firstly present the mathematical framework of FCA. Then, we show a way to transform a set of tuples into a binary relation which allows to characterize functional dependencies in that framework.

3.1 Formal Concept Analysis

Formal Concept Analysis (FCA) is a mathematical framework allowing to build a concept lattice from a binary relation between objects and their attributes. The concept lattice can be represented by a diagram where classes of objects/attributes and ordering relations between classes can be drawn, interpreted and used for data-mining, knowledge management and discovery [41, 42].

We use standard definitions from [17]. Let G and M be arbitrary sets and $I \subseteq G \times M$ be an arbitrary binary relation between G and M. The triple (G, M, I) is called a formal context. Each $g \in G$ is interpreted as an

object, each $m \in M$ is interpreted as an attribute. The statement $(g, m) \in I$ is interpreted as "g has attribute m". The two following derivation operators $(\cdot)'$:

$$A' = \{m \in M \mid \forall g \in A : gIm\} \qquad for\ A \subseteq G,$$
$$B' = \{g \in G \mid \forall m \in B : gIm\} \qquad for\ B \subseteq M$$

define a Galois connection between the powersets of G and M. The derivation operators $\{(\cdot)', (\cdot)'\}$ put in relation elements of the lattices $(\wp(G), \subseteq)$ of objects and $(\wp(M), \subseteq)$ of attributes and reciprocally. A Galois connection induces closure operators $(\cdot)''$ and realizes a one-to-one correspondence between all closed sets of objects and all closed sets of attributes. For $A \subseteq G$, $B \subseteq M$, a pair (A, B) such that $A' = B$ and $B' = A$, is called a *formal concept*. Concepts are partially ordered by $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ ($\Leftrightarrow B_2 \subseteq B_1$). (A_1, B_1) is a sub-concept of (A_2, B_2), while the latter is a super-concept of (A_1, B_1). With respect to this partial order, the set of all formal concepts forms a complete lattice called the *concept lattice* of the formal context (G, M, I), i.e. any subset of concepts has both a supremum (join $\vee$) and an infimum (meet $\wedge$) [17]. For a concept (A, B) the set A is called the *extent* and the set B the *intent* of the concept. The set of all concepts of a formal context (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$ while the concept lattice is denoted by $\underline{\mathfrak{B}}(G, M, I)$.

Theorem 1 (The Basic Theorem on Concept Lattices [17]) *The concept lattice of a context (G, M, I) is a complete lattice in which infimum and supremum are given by:*

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right)$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t \right)'', \bigcap_{t \in T} B_t \right)$$

An implication of a formal context (G, M, I) is denoted by $X \rightarrow Y$, $X, Y \subseteq M$ and means that all objects from G having the attributes in X also have also the attributes in Y, i.e. $X' \subseteq Y'$. Implications obey the Amstrong rules (reflexivity, augmentation, transitivity). A minimal subset of implications (in sense of its cardinality) from which all implications can be deduced with Amstrong rules is called the Duquenne-Guigues basis [19].

Objects described by non binary attributes can be represented in FCA as a many-valued context (G, M, W, I) with a set of objects G, a set of attributes M, a set of attribute values W and a ternary relation $I \subseteq G \times M \times W$. The statement $(g, m, w) \in I$, also written $g(m) = w$, means that "the value of attribute m taken by object g is w". The relation I verifies that $g(m) = w$ and $g(m) = v$ always implies $w = v$. For applying the FCA machinery, a many-valued context can be transformed into a formal context with a conceptual scaling. The choice of a scale should be wisely done w.r.t. data and goals since

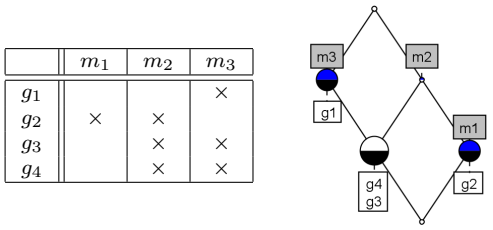

Fig. 1 A formal context and its concept lattice.

it affects the size, the interpretation, and the computation of the resulting concept lattice.

Example. Figure 1 shows a formal context and its concept lattice. Starting from an arbitrary set of objects, say $\{g_3\}$, one obtains concept $(\{g_3\}'', \{g_3\}') = (\{g_3, g_4\}, \{m_2, m_3\})$. The diagram shows the resulting concept lattice: each node denotes a concept while a line denotes an order relation between two concepts. The top (resp. bottom) concept is the highest (resp. lowest) concept w.r.t. the partial ordering of concepts ($\leq$).

Reduced labeling avoids to display the whole concept extents and intents. The extent of a concept has to be considered as composed of all objects attached to it and its sub-concepts; the intent of a concept is composed of all attributes attached to it and its super-concepts[1]. In this example, the implication $m_1 \rightarrow m_2$ holds, since $m_1' \subseteq m_2'$, i.e. $\{g_2\} \subseteq \{g_2, g_3, g_4\}$. Intuitively, implications can be read on the line diagram, as attributes labeling one concept implying attributes labeling itself or its super-concepts.

3.2 Functional Dependencies as Implications

We now recall with an example how functional dependencies can be characterized using FCA (see [4] and [17], page 92). The main idea behind this method consists in transforming a man-valued context into a formal context, whose concept lattice characterizes functional dependencies.

Starting from a tuple table T with attributes $\mathcal{U}$ taking values in Dom, we build the formal context $\mathbb{K} = (\mathcal{B}_2(G), M, I)$, where $G = T$ and $M = \mathcal{U}$ to respect the FCA notations from [17]. $\mathcal{B}_2(G) = \{ (t_i, t_j) \mid i < j \text{ and } t_i, t_j \in T \}$ is the set of pairs of tuples from G. Then, the relation I is defined as

$$(t_i, t_j)\ I\ m \Leftrightarrow t_i(m) = t_j(m), \text{ for } m \in M$$

This binary relation between pairs of tuples and attributes is reflexive, symmetric and transitive, and, therefore, it is an *equivalence relation*. The objects of $\mathbb{K}$ correspond to the set of all pairs of tuples from T (excluding symmetry and reflexivity to avoid redundancy), while attributes remain the

[1] The lattice drawing is done with the ConExp software: `conexp.sourceforge.net`

same. $((t_i, t_j), m) \in I$ means that the tuples t_i and t_j agree on the value taken by the attribute $m \in M$. Figure 2 illustrates the transformation of the initial data to build a formal context and its concept lattice. It should be noticed that the number of objects of the formal context is in the range of $O(|T^2|)$ (where $|T|$ is the number of tuples), so it can be significantly larger than the original set of tuples T.

id	a	b	c	d
t_1	1	3	4	1
t_2	4	3	4	3
t_3	1	8	4	1
t_4	4	3	7	3

$\mathbb{K}$	a	b	c	d
(t_1, t_2)		×	×	
(t_1, t_3)	×		×	×
(t_1, t_4)		×		
(t_2, t_3)			×	
(t_2, t_4)	×	×		×
(t_3, t_4)				

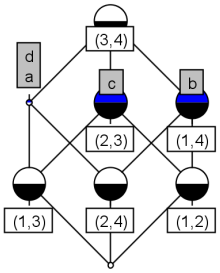

Fig. 2 Characterizing FDs with FCA: from a set of tuples to a formal context and its concept lattice.

We now explain how this concept lattice characterizes the set of all functional dependencies that hold in the table T with the following proposition:

Proposition 1 ([17,4]) *A functional dependency $X \to Y$ holds in a table T if and only if $\{X\}'' = \{X, Y\}''$ in the formal context $\mathbb{K} = (\mathcal{B}_2(G), M, I)$.*

This proposition states how to test that a FD holds using the concept lattice that has been computed. For instance, let us suppose that we want to test whether a functional dependency $a \to b$ holds in the formal context of Figure 2. We should test in the corresponding concept lattice if $\{a\}'' = \{a, b\}''$. In this particular case, we have that $\{a\}'' = \{a, d\}$ and $\{a, b\}'' = \{a, b, d\}$, which means that this dependency does not hold in T. On the other hand, the dependency $ac \to d$ holds, since $\{a, c\}'' = \{a, c, d\}$ and $\{a, c, d\}'' = \{a, c, d\}$.

An interesting consequence is that the set of implications that hold in the formal context $\mathbb{K} = (\mathcal{B}_2(G), M, I)$ is syntactically equivalent to the set of functional dependencies that hold in a table T [17,4]. By *syntactically* we mean that whenever an implication $X \to Y$ holds in $\mathbb{K}$, then the functional dependency $X \to Y$ holds in T (though not left-reduced). Equivalently, the minimal generating set of functional dependencies that hold in T is the same as the Duquenne-Guigues basis of the implications that hold in $\mathbb{K}$. Going back to our example, the concept lattice given in Figure 2 characterizes the implications $a \to d$ and $d \to a$, which form the Duquenne-Guigues basis.

3.3 Conceptual Scaling and FDs

Before introducing our method based on pattern structures to characterize functional dependencies, we investigate another aspect of the original data transformation into a formal context. In FCA, a way to turn a numerical table into a formal context is to use a conceptual scale (see Chapter 1.3 of [17]). Conceptual scaling consists in turning the many-valued attributes into binary attributes following rules given by the scale. For example, the ordinal scale states that, for a numerical attribute m, a pair object-attribute (g, m) taking value $x \in \mathbb{N}$ should be derived into binary attributes "$\leq y$", for any $y \geq x$ of the attribute domain, i.e. $(g, \text{“} \leq y\text{”}) \in I$. This means that the original dataset is turned into a formal context having the same set of objects and a larger set of binary attributes.

In the previous subsection, the data transformation we presented is not a conceptual scaling: the set of attributes remains the same after the transformation, whereas the set of objects is changed and its size is increased. Indeed, we replace objects by pairs of objects, and, given n objects, there are $n(n-1)/2$ potential pairs of objects. By contrast, we investigate in this section whether, given a data table T, it is possible to define a conceptual scaling applied to attributes and allowing to derive a formal context $\mathcal{K}_T$ with the same set of objects and such that the set of FDs holding in T is syntactically equivalent to the set of attribute implications holding in $\mathcal{K}_T$.

We show in the following example that this is not possible by constructing a simple and suitable counter-example. Let us consider the $n \times m$ numerical data table given in Figure 3 (left), based on $n = 4$ rows (objects) and $m = 4$ columns (attributes). Here the fact that $n = m$ here does not affect generality. The binarization, i.e. the transformation applied to objects (that could be termed as "vertical scaling"), yields $n(n-1)/2 = 6$ rows. The singularity of this example is that for any attribute, all objects share the same value except one (no empty row in the binary table), and this particular object is different for each attribute.

Actually, the context in Figure 3 (middle) is "clarified" and "reduced". Recall that a formal context (G, M, I) is clarified if $\forall g, h \in G$, $g' = h'$ implies $g = h$ (and similarly for the attributes). Moreover, an element x in a lattice L is $\vee$-irreducible (resp. $\wedge$-irreducible) if $x \neq \bot$ (resp. $x \neq \top$) and $x = a \vee b$ (resp. $x = a \wedge b$) implies $x = a$ or $x = b$ for all $a, b \in L$ [12]. Then in terms of FCA, a clarified context (G, M, I) is reduced when it is row-reduced (i.e. every object-concept is $\vee$-irreducible) and column-reduced (i.e. every attribute-concept is $\wedge$-irreducible) [17]. In addition, the number jir of $\vee$-irreducible concepts in a concept lattice is less than or equal to the number of objects $|G|$, and the number mir of $\wedge$-irreducible concepts is less than or equal to the number of attributes. There is equality when the formal context is clarified and reduced. For example, for the context given in Figure 3 (middle) and the associated concept lattice given in Figure 3 (right), we can observe that $mir = 4$ and $jir = 6$.

id	a	b	c	d
t_1	1	2	3	1
t_2	1	2	1	4
t_3	1	1	3	4
t_4	2	2	3	4

id	a	b	c	d
(t_1, t_2)	x	x		
(t_1, t_3)	x		x	
(t_1, t_4)		x	x	
(t_2, t_3)	x			x
(t_2, t_4)		x		x
(t_3, t_4)			x	x

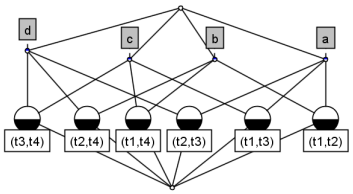

Fig. 3 A data table T (left) with its associated formal context $(\mathcal{B}_2(G), M, I)$ (middle). In the concept lattice diagram, nodes labeled with attributes (upper level) are $\wedge$-irreducible concepts while nodes labeled with objects (lower level) are $\vee$-irreducible concepts (right).

Now, scaling the data table T in Figure 3 (left) keeping unchanged the set of objects $G = T$ returns a formal context, say $(G, \hat{M}, \hat{I})$, where $|\hat{M}| \geq |M|$, i.e. the number of scaled attributes is greater than or equal to the initial number of attributes. Then, the number of $\wedge$-irreducible elements mir in the concept lattice $\underline{\mathfrak{B}}(G, \hat{M}, \hat{I})$ should verify $mir \geq 4$, as scaling separates attributes rather than merging them. In the same way, the number of $\vee$-irreducible elements jir in $\underline{\mathfrak{B}}(G, \hat{M}, \hat{I})$ should verify $jir \leq 4$. By contrast, $mir = 4$ and $jir = 6$ for the lattice $\underline{\mathfrak{B}}(\mathcal{B}_2(G), M, I)$. Then, it is not possible to find any scaling yielding a concept lattice $\underline{\mathfrak{B}}(G, \hat{M}, \hat{I})$ isomorphic to $\underline{\mathfrak{B}}(\mathcal{B}_2(G), M, I)$ –and thus with the same implication basis– as $\vee$-irreducible and $\wedge$-irreducible elements are preserved by the isomorphism. Thus, binarization should be necessarily applied to objects and not to attributes.

4 Characterizing FDs with Pattern Structures

In the previous section, we showed how to turn a set of tuples T into a formal context $\mathbb{K} = (\mathcal{B}_2(G), M, I)$, whose concept lattice allows to characterize functional dependencies. However, the number of objects $|\mathcal{B}_2(G)|$ in the resulting context is quadratic with respect to the number of tuples. As shown later in the experiments, this is not viable for real datasets. Thus, we propose to use the formalism of pattern structures to obtain an equivalent concept lattice, avoiding a transformation leading to a quadratic number of objects. Pattern structures can be understood as a generalization of FCA able to directly deal with complex data i.e. objects taking descriptions in a partially ordered set.

4.1 Pattern Structures

A pattern structure is defined as a generalization of a formal context describing complex data [16]. Formally, let G be a set of objects, let $(D, \sqcap)$ be a meet-semi-lattice of potential object descriptions and let $\delta : G \longrightarrow D$ be a mapping associating each object with its description. Then $(G, (D, \sqcap), \delta)$ is a pattern structure. Elements of D are patterns and are ordered thanks to a subsumption relation $\sqsubseteq$: $\forall c, d \in D$, $c \sqsubseteq d \iff c \sqcap d = c$.

A pattern structure $(G, (D, \sqcap), \delta)$ is based on two derivation operators $(\cdot)^{\square}$:

$$A^{\square} = \prod_{g \in A} \delta(g) \qquad for\ A \subseteq G$$
$$d^{\square} = \{g \in G | d \sqsubseteq \delta(g)\} \qquad for\ d \in (D, \sqcap).$$

These operators form a Galois connection between $(\wp(G), \subseteq)$ and $(D, \sqcap)$. Pattern concepts of $(G, (D, \sqcap), \delta)$ are pairs of the form (A, d), $A \subseteq G$, $d \in (D, \sqcap)$, such that $A^{\square} = d$ and $A = d^{\square}$. For a pattern concept (A, d), d is a pattern intent and is the common description of all objects in A, the pattern extent. When partially ordered by $(A_1, d_1) \leq (A_2, d_2) \Leftrightarrow A_1 \subseteq A_2$ $(\Leftrightarrow d_2 \sqsubseteq d_1)$, the set of all concepts forms a complete lattice called pattern concept lattice.

As for formal contexts, implications can be defined. For $c, d \in D$, the pattern implication $c \rightarrow d$ holds if $c^{\square} \subseteq d^{\square}$, i.e. the pattern d occurs in an object description if the pattern c does. Similarly, for $A, B \subseteq G$, the object implication $A \rightarrow B$ holds if $A^{\square} \sqsubseteq B^{\square}$, meaning that all patterns that occur in all objects from the set A also occur in all objects in the set B [16].

Finally, it can be noticed that existing FCA algorithms [26] can be reused with slight modifications to compute pattern structures, in order to extract and classify concepts [22].

4.2 The Partition Lattice as a Space of Descriptions

In order to construct the meet-semi-lattice of potential object descriptions of a pattern structure, we recall well-known definitions of the partitions of a set and the so-called partition lattice. In the examples that follow, we consider a set $E = \{1, 2, 3, 4\}$.

Partition of a set. A partition over a given set E is a set $P \subseteq \wp(E)$ s.t.:

- $\bigcup_{p_i \in P} p_i = E$
- $p_i \cap p_j = \emptyset$, for any $p_i, p_j \in P$ with $i \neq j$.

In other words, a partition covers E and is composed of disjoint subsets of E.

Equivalence relation. There is a bijection between the sets of partitions and the set of equivalence relations of a set. This 1-1-correspondence between P and R_P is given by $(e, e') \in R_P$ iff e and e' belongs to the same equivalence class of P [11,18]. For example, given $P = \{\{1, 2, 3\}, \{4\}\}$, one has the relation $R_P = \{(1, 2), (1, 3), (2, 3), (1, 1), (2, 2), (3, 3), (4, 4)\}$ (omitting symmetry for the sake of readability).

The set of equivalence relations on any set T can be ordered by inclusion, if we consider a relation as a set of pairs of T, or also as the natural order on partitions, if we take the partition notation for the relations.

Ordering relation. A partition P_1 is finer than a partition P_2 (P_2 is coarser than P_1), written $P_1 \sqsubseteq P_2$ if any subset of P_1 is a subset of a subset in P_2. For example,

$$\{\{1,3\},\{2\},\{4\}\} \sqsubseteq \{\{1,2,3\},\{4\}\}$$

In fact, the sets of equivalence relations of a set T, or the set of partitions of T, is a lattice. The unit element (top) of this lattice denotes the fact that all objects are equivalent, i.e. all the attributes are in one class, while in the zero element (bottom) there are no two equivalent elements, i.e. each single element forms an equivalence class ($|T|$ classes of equivalence). Seen as sets of pairs $T \times T$, the top element contains precisely $T \times T$, whereas the zero element contains the sets $\{\,(x,x) \mid \forall x \in T\,\}$.

We can define the meet of two equivalence relations, or two partitions, as follows:

Meet of two partitions. It is defined as the coarsest common refinement of two partitions. In other words, it is the intersection of the respective equivalence relations (omitting reflexivity for the sake of readability):

$$\{\{1,3\},\{2,4\}\} \sqcap \{\{1,2,3\},\{4\}\} = \{\{1,3\},\{2\},\{4\}\}$$
$$\text{or } \{(1,3),(2,4)\} \cap \{(1,2),(1,3),(2,3)\}$$

The meet of two partitions is identical to the intersection of two equivalence relations seen as sets of pairs of tuples. Likewise, we can also define the join of two equivalence relations or partitions, which, again, can be seen as the union of two sets of pairs:

Join of two partitions. It is defined as the finest common coarsening of two partitions. In other words, it is the transitive closure of the union of the respective equivalence relations.

$$\{\{1,3\},\{2\},\{4\}\} \sqcup \{\{1,2\},\{3\},\{4\}\} = \{\{1,2,3\},\{4\}\}$$
$$\text{or } \mathit{transitive\_closure}(\{(1,3)\} \cup \{(1,2)\}) = \{(1,2),(1,3),(2,3)\}$$

Finally, one should notice that the property $P_1 \sqcap P_2 = P_1 \Leftrightarrow P_1 \sqsubseteq P_2$ naturally holds (and the dual for join). Since the set of all partitions over a set forms a lattice $(D, \sqcap, \sqcup)$, it can be used as a description space of a pattern structure.

4.3 Partition Pattern Structure

Consider a tuple table T as a many-valued context (G, M, W, J) where $G = T$ corresponds to the set of objects ("rows"), $M = \mathcal{U}$ to the set of attributes ("columns"), $W = Dom$ the data domain ("all distinct values of the table") and $J \subseteq G \times M \times W$ a relation such that $(g, m, w) \in J$ also written $m(g) = w$ means that attribute m takes the value w for the object g [17]. In Table 4 (left), we have $d(t_4) = 3$.

id	a	b	c	d
t_1	1	3	4	1
t_2	4	3	4	3
t_3	1	8	4	1
t_4	4	3	7	3

m	$\delta(m) \in (D, \sqcap)$
a	$\{\{t_1, t_3\}, \{t_2, t_4\}\}$
b	$\{\{t_1, t_2, t_4\}, \{t_3\}\}$
c	$\{\{t_1, t_2, t_3\}, \{t_4\}\}$
d	$\{\{t_1, t_3\}, \{t_2, t_4\}\}$

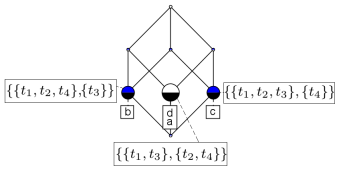

Fig. 4 The original data (left), the resulting pattern structure (middle) and its pattern concept lattice (right)

We show how a partition pattern structure can be defined from a many-valued context (G, M, W, J) and show that its concept lattice is equivalent to the concept lattice of $\mathbb{K} = (\mathcal{B}_2(G), M, I)$ introduced above. Intuitively, formal objects of the pattern structure are the attributes of the many-valued context (G, M, W, J). Then, given an attribute $m \in M$, its description $\delta(m)$ is given by a partition over G such that any two elements g, h of the same class take the same values for the attribute m, i.e. $m(g) = m(h)$. The result is given in Figure 4 (middle). As such, descriptions obey the ordering of a partition lattice as described above. It follows that (G, M, W, J) can be represented as a pattern structure $(M, (D, \sqcap), \delta)$ where M is the set of original attributes, and $(D, \sqcap)$ is the set of partitions over G provided with the partition intersection operation $\sqcap$. An example of concept formation is given as follows, starting from set $\{a, d\} \subseteq M$:

$$\begin{aligned} \{a, d\}^{\square} &= \delta(a) \sqcap \delta(d) \\ &= \{\{t_1, t_3\}, \{t_2, t_4\}\} \sqcap \{\{t_1, t_3\}, \{t_2, t_4\}\} \\ &= \{\{t_1, t_3\}, \{t_2, t_4\}\} \\ \{\{t_1, t_3\}, \{t_2, t_4\}\}^{\square} &= \{m \in M | \{\{t_1, t_3\}, \{t_2, t_4\} \sqsubseteq \delta(m)\} \\ &= \{a, d\} \end{aligned}$$

Hence, $(\{a, d\}, \{\{t_1, t_3\}, \{t_2, t_4\}\})$ is a pattern concept. The resulting pattern concept lattice is given in Figure 4 (right).

In the previous section, a many-valued context (G, M, W, J) was derived as the formal context $(\mathcal{B}_2(G), M, I)$ where $\mathcal{B}_2(G)$ represents any pair of objects, and $((g, h), m) \in I$ means that $m(g) = m(h)$. The resulting concept lattice is used to characterize the set of FDs [17]. A new result is that both structures $(\mathcal{B}_2(G), M, I)$ and $(M, (D, \sqcap), \delta)$ are equivalent, i.e. both collections of concepts are in 1-1-correspondence.

Proposition 2 *(B, A) is a pattern concept of the partition pattern structure $(M, (D, \sqcap), \delta)$ if and only if (A, B) is a formal concept of the formal context $(\mathcal{B}_2(G), M, I)$ for all $B \subseteq M, A \subseteq \mathcal{B}_2(G)$ (equivalently A is a partition on G).*

Proof We first notice that a pattern $A \in D$ is a partition of the set of tuples, whereas elements of $\mathcal{B}_2(G)$ are sets of pairs of tuples. Yet, as we have seen in

Subsection 3.2, objects in $\mathcal{B}_2(G)$ form an equivalence relation, and, therefore, partitions, which means that they correspond to patterns in D.

Consider now that the concept lattices of the contexts $(\mathcal{B}_2(G), M, I)$ and $(M, \mathcal{B}_2(G), I)$ are equivalent, as they are built with "symmetric concepts": if (A, B) belongs to the first, (B, A) belongs to the second. With the context $(M, \mathcal{B}_2(G), I)$ and the pattern structure $(M, (D, \sqcap), \delta)$, the proposition holds since $B' = B^{\square}$ for all $B \subseteq M$:

$$\begin{aligned} B^{\square} &= \prod_{m \in B} \delta(m) \\ &= \bigcap_{m \in B} \{ (t, t') \mid t(m) = t'(m) \} \quad \forall t, t' \in T \\ &= \{ (t, t') \mid t(B) = t'(B) \} \quad \forall t, t' \in T \\ &= B' \end{aligned}$$

And symmetrically, $A' = A^{\square}$ for all $A \in D$, $A \subseteq \mathcal{B}_2(G)$:

$$\begin{aligned} A^{\square} &= \{m \in M \mid A \sqsubseteq \delta(m)\} \\ &= \{m \in M \mid \forall (t, t') \in A : (t, t') \in \delta(m)\} \quad \forall t, t' \in T \\ &= \{m \in M \mid \forall (t, t') \in A : t(m) = t'(m)\} \quad \forall t, t' \in T \\ &= A' \end{aligned}$$

Example. The pattern concept $(\{b\}, \{\{1, 2, 4\}, \{3\}\})$ is equivalent to the formal concept $(\{(1, 2), (1, 4), (2, 4)\}, \{b\})$. One should remark that pattern structures offer more concise intent representation when the set of tuples becomes very large, i.e. storing a partition instead of all pairs of objects that are together in a same class of the partition.

Moreover, there is an isomorphism between the concept lattice of (G, M, I) and the pattern concept lattice of $(G, (D, \sqcap), \delta)$. Then, the following proposition states that FDs can be characterized within the pattern concept lattice.

Proposition 3 *A functional dependency $X \to Y$ holds in a table T if and only if: $\{X\}^{\square} = \{XY\}^{\square}$ in the partition pattern structure $(M, (D, \sqcap), \delta)$.*

Proof First of all, we notice that $(t, t') \in X^{\square}$ if and only if $t(X) = t'(X)$, i.e. $\forall x \in X : t(x) = t'(x)$. We also notice that $\{X, Y\}^{\square} \subseteq \{X\}^{\square}$, as $\{X\} \subseteq \{X, Y\}$.
($\Rightarrow$) We prove that if $X \to Y$ holds in T, then, $\{X\}^{\square} = \{X, Y\}^{\square}$, i.e. $\{X\}^{\square} \subseteq \{X, Y\}^{\square}$. We take an arbitrary pair $(t, t') \in \{X\}^{\square}$, i.e. $t(X) = t'(X)$. Since $X \to Y$ holds, it implies that $t(XY) = t'(XY)$, and this implies that $(t, t') \in \{X, Y\}^{\square}$.
($\Leftarrow$) We take an arbitrary pair $t, t' \in T$ such that $t(X) = t'(X)$. Therefore, we have that $(t, t') \in X^{\square}$, and by hypothesis, $(t, t') \in XY^{\square}$, i.e. $t(XY) = t'(XY)$. Since this is true for all pairs $t, t' \in T$ such that $t(X) = t'(X)$, it comes that $X \to Y$ holds in T.

Example. We consider a FD that holds in Table 1: $a \to d$. It is characterized from $(\mathcal{B}_2(G), M, I)$ as an attribute implication. It holds as well in $(M, \mathcal{B}_2(G), I)$ and $(M, (D, \sqcap), \delta)$ as an object implication.

Therefore, a naive algorithm that computes $\{ X \to XY \mid \{X\}^{\square} = \{XY\}^{\square} \}$ for all $X, Y \subseteq \mathcal{U}$ would compute the Functional Dependencies that hold in a table. Alternatives are discussed in section 6 and 7.

5 Characterizing DMVDs with Pattern Structures

In order to show the flexibility of pattern structures to characterize dependencies, we now introduce how to handle a more general type of dependencies: degenerated multivalued dependencies (DMVDs). We have seen that the computation of functional dependencies is based on the equivalence relations (partitions) that are induced by an attribute. In order to compute DMVDs we consider now a *tolerance relation*. This relation is different from an equivalence relation in that it is symmetric and reflexive but not necessarily transitive. We define a tolerance relation on the set of tuples of a relation induced by an attribute:

Definition 3 Let $a \in \mathcal{U}$ and let $\overline{a} = \mathcal{U} \setminus \{ a \}$. The tolerance relation $\mathcal{R}_{\mathrm{T}}$ in a table T induced by a is:

$$\mathcal{R}_{\mathrm{T}}(a) = \{(t_i, t_j) \in T \times T \mid i < j \text{ and } t_i(a) = t_j(a) \text{ or } t_i(\overline{a}) = t_j(\overline{a})\}$$

With the restriction $i < j$ we prevent pairs such as (t_i, t_i), or two symmetric pairs (t_i, t_j) and (t_j, t_i) from appearing in the representation of a relation, because, since symmetry and reflexivity hold, their presence is redundant. Note that the difference w.r.t. the definition of functional dependencies is the addition of the conjunctive clause $t_i(\overline{a}) = t_j(\overline{a})$.

Example: Consider the example of the table on the right. We see that $\mathcal{R}_{\mathrm{T}}(a) = \{(t_1, t_2), (t_2, t_3)\}$, (reflexivity and symmetry are omitted) but $(t_1, t_3) \notin \mathcal{R}_{\mathrm{T}}(a)$, which would hold because of transitivity.

id	a	b	c	d
t_1	1	1	1	1
t_2	1	2	2	2
t_3	3	2	2	2

Since tolerance relations are sets of pairs of tuples, if we define the meet and join between two tolerance relations as their set intersection and union, and order them by set inclusion, we have that the set of all possible tolerance relations is a complete lattice.

Given a tolerance relation, so called blocks of tolerance are defined as maximal sets of pairwise elements in correspondence (see [25] in FCA settings):

Definition 4 Given a set G, a subset $K \subseteq G$, and a tolerance relation I on G, K is a *block of tolerance* if:

(i) $\forall x, y \in K \; xIy$ (pairwise in correspondence)
(ii) $\forall z \notin K, \exists u \in K \; \neg(zIu)$ (maximality)

For instance, the tolerance block $\{t_1, t_2, t_4\}$ represents the set of pairs $\{(t_1, t_2), (t_1, t_4), (t_2, t_4)\}$. An attribute $m \in M$ is no longer described by a partition over the set of objects as in the previous section, but rather by a set of tolerance blocks. The pattern structure that we obtain, denoted by $(M, (D, \sqcap), \delta)$, is such that $\delta(m)$ maps an attribute $m \in M$ to the set of tolerance blocks of the relation $\mathcal{R}_{\mathrm{T}}(m)$. The description space $(D, \sqcap)$ admits the same meet $\sqcap$ and the same ordering relation $\sqsubseteq$ as the partition lattice. Then, an example of concept formation is given as follows, starting from the set $\{a, b\} \subseteq M$:

$$\begin{aligned}
\{a,b\}^{\square} &= \delta(a) \sqcap \delta(b) \\
&= \{\{t_1,t_3\},\{t_2,t_4\}\} \sqcap \{\{t_1,t_2,t_4\},\{t_1,t_3\}\} \\
&= \{\{t_1,t_3\},\{t_2,t_4\}\} \\
\{\{t_1,t_3\},\{t_2,t_4\}\}^{\square} &= \{m \in M \mid \{\{t_1,t_3\},\{t_2,t_4\}\} \sqsubseteq \delta(m)\} \\
&= \{a,b,c,d\}
\end{aligned}$$

The resulting pattern structure along with its pattern concept lattice is given in Figure 5.

$m \in M$	$\delta(m) \in (D, \sqcap)$
a	$\{\{t_1,t_3\},\{t_2,t_4\}\}$
b	$\{\{t_1,t_2,t_4\},\{t_1,t_3\}\}$
c	$\{\{t_1,t_2,t_3\},\{t_2,t_4\}\}$
d	$\{\{t_1,t_3\},\{t_2,t_4\})\}$

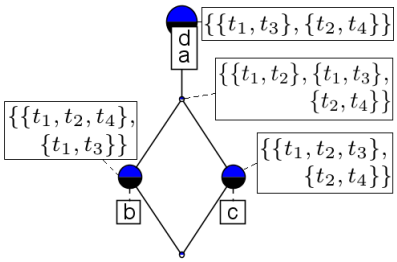

Fig. 5 Characterizing DMVDs with a pattern concept lattice: The pattern structure (on the left) obtained by transforming Table 1 and its pattern concept lattice (on the right)

$\mathbb{K}$	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
a		×			×	
b	×	×	×		×	
c	×	×		×	×	
d		×			×	

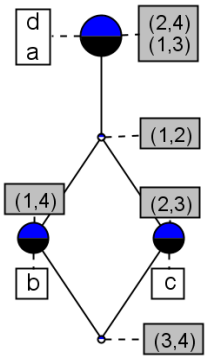

Fig. 6 Characterizing DMVDs with FCA.

It can be noticed that, as for functional dependencies, a formal context can be built to characterize DMVDs, whose concept lattice is equivalent. The formal context can be written as $(M, \mathcal{B}_2(G), R)$ where $(m, (t_i, t_j)) \in R \iff (t_i, t_j) \in \mathcal{R}_{\mathrm{T}}(m)$. The resulting formal context of our example and its concept lattice are given in Figure 6. Here again, pattern structures offer more concise object descriptions with sets of blocks of tolerance instead of sets of pairs of tuples.

Now we can state how a DMVD $X \to Y$ holds in T according to the pattern structure $(M, (D, \sqcap), \delta)$.

Theorem 2 *Let $Z = \mathcal{U} \setminus X \setminus Y$. A DMVD $X \to Y$ holds in T if and only if*

$$\{X\}^{\square} = \{XY\}^{\square} \cup \{XZ\}^{\square}$$

Proof We assume that $(t, t') \in X^{\square}$ and $X' \subseteq X$ implies that $(t, t') \in X'^{\square}$. We also have that $Z = \mathcal{U} \setminus X \setminus Y$.
($\Rightarrow$) We take two different tuples $t, t' \in X^{\square}$. We have two different options:

1. $t(X) \neq t'(X)$. This implies, necessarily, that there is a subset of attributes $W \subseteq X$ such that $t(W) \neq t'(W)$, which implies that $t(\overline{W}) = t'(\overline{W})$. In this case, since $YZ \subseteq \overline{X}$ we have that for all $x \in YZ : (t, t') \in \delta(x)$, and, therefore, $(t, t') \in XYZ^{\square}$, i.e. $(t, t') \in \{XY\}^{\square} \cup \{XZ\}^{\square}$.
2. $t(X) = t'(X)$. Since $X \to Y$ holds in T, we have that $t(Y) = t'(Y)$ or $t(Z) = t'(Z)$. In the first case, we have that, for all $y \in Y : t(y) = t'(y)$, and then, $(t, t') \in \delta(y)$. This yields that $(t, t') \in XY^{\square}$. The case $t(Z) = t'(Z)$ is symmetric.

In both cases we have that $(t, t') \in \{XY\}^{\square} \cup \{XZ\}^{\square}$.

($\Leftarrow$) We take two different tuples (t, t') such that $t(X) = t'(X)$. This implies that $(t, t') \in X^{\square}$, and, therefore, by hypothesis, that $(t, t') \in XY^{\square}$ or $(t, t') \in XZ^{\square}$. Both cases are symmetric, and we take the former one: $(t, t') \in XY^{\square}$, and in this case we have two different cases:

1. There is a subset of attributes $W \in Y$ such that $t(W) \neq t'(W)$. In this case, we have that necessarily $t(\overline{W}) = t'(\overline{W})$. Since $Z \subseteq \overline{Y}$, then, $t(Z) = t'(Z)$ and the DMVD $X \to Y$ holds in T (by symmetry).
2. We have that $t(Y) = t'(Y)$, in which case the DMVD $X \to Y$ holds in T.

We find here a method to check whether a DMVD holds in a table, similar to that described in Section 3. For instance, if we want to check if $a \to b$ we check if $a^{\square} = ab^{\square} \cup acd^{\square}$, which is true since $a^{\square} = \{(t_1, t_3), (t_2, t_4)\}$, $ab^{\square} = \{(t_1, t_3), (t_2, t_4)\}$ and $acd^{\square} = \{(t_1, t_3), (t_2, t_4)\}$. If we want to test whether $c \to a$, we see that $c^{\square} = \{(t_1, t_2), (t_1, t_3), (t_2, t_3), (t_2, t_4)\}$ whereas $ac^{\square} = \{(t_1, t_3), (t_2, t_4)\}$ and $bcd^{\square} = \{(t_1, t_3), (t_2, t_4)\}$ which implies that $c^{\square} \neq ac^{\square} \cup bcd^{\square}$, and by Theorem 2 means that $c \to a$ does not hold in T.

As we did in the previous section, we do not discuss how to enumerate all the DMVDs that hold in a table, although Theorem 2 states that a naive algorithm that computes $\{ X \to XY \mid \{X\}^{\square} = \{XY\}^{\square} \cup \{XZ\}^{\square} \}$ for all $X, Y \subseteq \mathcal{U}$ would be enough.

6 Experiments

We showed how pattern structures can alternatively represent the formal context $(M, \mathcal{B}_2(G), I)$, or equivalently $(\mathcal{B}_2(G), M, I)$, by means of partition patterns. Both concept lattices are equivalent and thus can be used to characterize FDs. To assess the usefulness of introducing partition pattern structures, we applied both methods to well known UCI datasets[2].

To compute with formal contexts, we wrote a simple procedure to transform a many-valued context (G, M, W, J), or table T, into a formal context, and applied the (C++) closed itemset mining algorithm LCM (version 2 [40]). Whereas this algorithm only computes concept intents, it is known to be one of the most efficient for that task. We also consider clarified contexts $(\mathcal{B}_2(G), M, I)$: clarifying the objects of a context consists in keeping only one object among those that have the same closure. It is known that both original, say non-clarified, and clarified contexts give rise to equivalent concept lattices that hold the same implications, hence the same functional dependencies. However, the number of objects of the clarified context can be smaller in several orders of magnitude.

To compute with pattern structures, we turned a many-valued context into a set of partitions over G (one for each attribute $m \in M$) and applied a slight (Java) modification of the algorithm CloseByOne [26]. Indeed, the latter can be easily adapted by changing the definition of both intersection and subsumption test, used for closure computation (a detailed explanation for another instance of pattern structures can be found in [22,23]). As such, this method computes pattern concepts, i.e. both pattern extents and intents.

Table 2 gives the details of the datasets and their derived formal contexts. It can be noticed that in column $|\mathcal{B}_2(G)|$, formal objects (g, h) with empty description, i.e. $\{(g, h)\}' = \emptyset$ for any $g, h \in G$, are not taken into account. The same applies for the last column (right-most) where we count the number of unique non-empty object descriptions only, i.e. after clarifying the objects. It can be noticed that the number of objects after clarification is much smaller (less than 3% in average on all datasets).

Table 3 gives the execution time of both methods. For pattern structures, execution times include the reading of the data, their process to a set of partitions and the CloseByOne execution. Concerning formal contexts, we evaluate the performances of LCM without and with object clarification: (i) execution times include data reading and process with LCM while the time to build the formal context is not taken into account, (ii) clarification and LCM processing are monitored separately.

[2] `http://archive.ics.uci.edu/ml/datasets.html`

In both cases, algorithms only output the number of patterns. The experiments were carried out on an Intel Core i7 CPU 2.40 Ghz machine with 4 GB RAM.

	(G, M, W, I)		$(\mathcal{B}_2(G), MI)$			
Dataset	$\|G\|$	$\|M\|$	$\|\mathcal{B}_2(G)\|$	Avg. $\|g'\|$	Density	$\|\mathcal{B}_2(G)\|$clarified
iris	150	5	4,3K	1.38	27%	26
hepatitis	155	20	11K	9.02	45%	6,071
glass	214	10	19K	1.74	17%	105
imports-85	205	26	20K	6.24	24%	2,767
balance-scale	625	5	143K	1.67	33%	29
crx	690	16	236K	5.53	43%	4,398
flare	1,066	13	567K	8.79	67%	1,551
abalone	4,177	9	3,7M	1.19	13%	240
krkopt-25%	7,013	7	20M	1.84	26%	125
krkopt-50%	14,027	7	76M	1.72	24%	125
krkopt-75%	21,040	7	171M	1.66	24%	125
krkopt-100%	28,056	7	299M	1.67	23%	125
adult-25%	8,140	14	33M	6.32	42%	7,795
adult-50%	16,280	14	132M	6.34	42%	8,709
adult-75%	24,320	14	295M	6.34	42%	9,192
adult-100%	32,561	14	530M	6.33	42%	9,554

Table 2 Datasets and their characteristics (K stands four thousands, M for millions)

Dataset	Number of intents	Time CloseByOne	Time LCM no clar.	Time Clarification	Time LCM after clar.
iris	26	≤ 1	≤ 1	≤ 1	≤ 1
balance-scale	30	≤ 1	≤ 1	≤ 1	≤ 1
flare	4,096	≤ 1	≤ 1	≤ 1	≤ 1
glass	133	≤ 1	≤ 1	≤ 1	≤ 1
crx	9,528	4	≤ 1	≤ 1	≤ 1
abalone	252	5	≤ 1	≤ 1	≤ 1
hepatitis	95,576	11	≤ 1	≤ 1	≤ 1
imports85	205,623	228	≤ 1	≤ 1	≤ 1
krkopt-25%	126	$\leq$ **1**	6	4	≤ 1
krkopt-50%	126	$\leq$ **1**	N/A	16	≤ 1
krkopt-75%	126	$\leq$ **1**	N/A	36	≤ 1
krkopt-100%	126	$\leq$ **1**	N/A	64	≤ 1
adult-25%	10,881	**4**	N/A	24	≤ 1
adult-50%	12,398	**5**	N/A	95	≤ 1
adult-75%	13,133	**10**	N/A	213	≤ 1
adult-100%	13,356	**12**	N/A	414	≤ 1

Table 3 Comparing pattern structures and formal context representations. Execution times are given in seconds. N/A means that the computation was intractable for memory issues.

From Table 3, it can be observed than computing with formal contexts is faster for the smallest datasets, even *abalone* that holds more than 3 millions

of objects. However, with bigger datasets, from 20 to 530 millions of objects, partition pattern structures are the only method able to compute the set of concepts. This holds for 7 numerical attributes already, and is accentuated with 15. It is indeed already known that complexity of computing FDs is highly related to the number of numerical attributes M.

Now, as we noticed before, after the object clarification it remains a very small proportion of objects, hence leading to a very fast computation with LCM, which outperforms CloseByOne on pattern structures. However, the time required for clarifying the context makes CloseByOne still an efficient alternative. Indeed, one still needs to generate $|G| \times |G|$ object descriptions and to keep the unique object descriptions. As such, the scaling processes each pair of objects, i.e. builds its description, and the later is added in a sorted set data-structure (ensuring $log(n)$ time cost, hence $n^2 log(n)$ in total). When all pairs have been considered, the resulting clarified formal context is processed with LCM which runs faster on a reduced set of objects.

As already suggested in [22,16] in different settings, the trade-off of performances between the process of formal contexts and pattern structures is explained as follows. When working with simple descriptions (i.e. vectors of bits), computing an intersection is more efficient than when working with more complex descriptions. Indeed, partitions are encoded in our algorithm as vectors of bitvectors (i.e. partitions) and both intersections or inclusion test computation require to consider all pairs of sets between the two partitions in argument. Although we used optimizations avoiding an exhaustive computation between all pairs (by considering a lectic order on parts), these operations are more complex than standard intersections and inclusion tests between sets. However, we need to compute much less intersections, thus the following trade-off. Pattern structures perform better with larger datasets. Formal objects (numerical attributes) are mapped into concise descriptions (partitions) whereas they are mapped with the equivalence class of the same partitions in the case of formal contexts. Consequently, pattern structures are preferred to formal contexts when the number of possible pairs of objects that agree for one or more attributes is high ($|\mathcal{B}_2(G)|$).

7 Related Work

Dependency theory is an important subject of database theory for more than twenty years. Several types of dependencies have been proposed, capturing different semantics, and useful for different tasks among which query optimization, normalization, data cleaning, error detection, etc. We draw attention on functional dependencies and degenerated multi-valued dependencies in the present article, while several works studied also equality generating dependencies (that generalize FDs) [8], constraint generating dependencies (where equality is replaced by others constraints) [6] or closely related, differential dependencies [36], conditional functional dependencies (that hold in instances of the relation) [10,30,29,13], association rules (that hold on particular values

of attributes) [2], matching dependencies and dependencies in fuzzy settings (to cope with uncertainty) [14,37,9], etc.

Most of the existing algorithms allowing to discover dependencies from an arbitrary relation rely on a level-wise exploration of the attribute set lattice [2]. For example, TANE is an algorithm for computing functional dependencies [20]. It performs a bottom-up exploration of the attribute lattice, combined with a pruning strategy applied when computing a new level. For each set of attributes, it computes the partition associated to each set $X \subseteq \mathcal{U}$. This partition is computed as a product of two previously computed partitions. This *product* stands for the intersection of computed partitions seen as set of pairs, that is, as an equivalence relation. The difference with respect to our approach (i.e. computing a pattern concept lattice) is that instead of keeping the sets that compose a pattern concept lattice, the TANE algorithm keeps only the set of minimal functional dependencies. Yet, in order to compute those dependencies, it is needed to compute the partitions for the required sets of attributes. Therefore, even if the output is not the same (sets of dependencies instead of a pattern concept lattice), the result is equivalent. As such, it is not fair to compare our pattern structure algorithm with TANE. However, we performed a few experiments on the same benchmark dataset, showing that pattern structures form a good candidate for a further investigation on computing functional dependencies, and also other kind of dependencies.

Depth-first approaches have also been considered, e.g. with the heuristic driven approach realized by the algorithm FastFDs [43]. Finally, to deal with the issues of very large heterogeneous databases and uncertainty, approximate approaches (greedy and randomized approaches with approximation bounds on errors) are also developed [38].

The characterization of functional dependencies with FCA has been dealt with in [4,30,29] and in [17], as it was explained in Section 3. [29] shows that association rules (ARs), functional dependencies (FDs), and conditional functional dependencies (CFDs) follow a hierarchy: FDs are the union of CFDs, the latter are the union of ARs. This work is extended in [30] where a lattice characterization of CFDs is proposed. We address a comparison with this formalism for CFDs in perspectives. FCA allows also to draw complexity results on dependency theory, e.g. recently, where the problem of recognizing whether a subset of attributes is a premise of a minimal cover of functional dependencies of a relation is shown to be coNP-complete [3].

An FCA characterization of DMVDs is presented and discussed in [5]. Other more sophisticated dependencies, such as multivalued dependencies and acyclic join dependencies are dealt with in [4], The characterization consists in the creation of a formal context (G, M, I) such that G is formed by combining tuples of the original table, and M is formed also by combining the original attribute set of the table. The process is similar to the one described in Section 3, i.e. no implicit transformation of the original data was performed, but the size of the resulting context is proportional to the size of the original data, leading to the same problem found in computing FDs with FCA.

Defining a concept lattice where objects take their descriptions in the lattice of partitions of a given set has also introduced in [31]. After introducing so called *agree concept lattice*, the authors highlight its possible usage for the discovery of functional dependencies, but also to tackle the problem of skyline computation in databases (the notion of agree sets was introduced in [7]). This structure is equivalent to the partition concept lattice. Indeed, working with partitions has been early identified as a key element for the computing of functional dependencies (see e.g. [27,20]). The main difference in our work is to show that pattern structures can be directly applied, and one does not need to prove that $(.)^{\square\square}$ is a closure operator (Galois connection). Hence, pattern structures appear to be a flexible way to handle dependencies (as we showed for handling FDs and DMVDs).

In this article, we consider the problem of characterizing FDs with implications in FCA. In [24], the inverse reduction was given: For a context $\mathbb{K} = (G, M, I)$ one can construct a many-valued context (relational table) K_W such that an implication $X \to Y$ holds iff Y is functionally dependent on X in K_W.

8 Conclusion

On one hand, the discovery of functional dependencies is an important topic in the field of databases. On the other hand, the discovery of implications is an attracting topic in formal concept analysis. We started our investigation from a known result that links both fields: functional dependencies can be characterized with formal concept analysis after a data transformation leading to a heavy data representation. Accordingly, we tackled the problem of avoiding such transformation by introducing partition pattern structures, a new conceptual structure that allows an equivalent characterization, but coming with better computational properties. Indeed, the empirical results show that, although the classical FCA approach performs well for small datasets, it is not scalable compared to partition pattern structures. Since real-world datasets become larger and larger, this scalability is a more important feature than the speed concern for small datasets.

We think that the above results, on the formalization and the computation of dependencies, open the possibility to adapt pattern structures to other kinds of dependencies, namely, multi-valued dependencies and similar constraints that may be found in different fields.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading (MA), USA, 1995.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

3. M. A. Babin and S. O. Kuznetsov. Computing premises of a minimal cover of functional dependencies is intractable. *Discrete Applied Mathematics*, 161(6):742–749, 2013.
4. J. Baixeries. *Lattice Characterization of Armstrong and Symmetric Dependencies (PhD Thesis)*. Universitat Politècnica de Catalunya, 2007.
5. J. Baixeries and J. L. Balcázar. Characterization and armstrong relations for degenerate multivalued dependencies using formal concept analysis. In B. Ganter and R. Godin, editors, *ICFCA*, volume 3403 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2005.
6. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.
7. C. Beeri, M. Dowd, R. Fagin, and R. Statman. On the structure of armstrong relations for functional dependencies. *Journal of the ACM*, 31(1):30–46, 1984.
8. C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13(1):76–98, 1984.
9. R. Belohlávek and V. Vychodil. Data tables with similarity relations: Functional dependencies, complete rules and non-redundant bases. In M.-L. Lee, K.-L. Tan, and V. Wuwongse, editors, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 644–658. Springer, 2006.
10. P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis, editors, *ICDE*, pages 746–755. IEEE, 2007.
11. N. Caspard and B. Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.
12. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, 1990.
13. T. Diallo, N. Novelli, and J.-M. Petit. Discovering (frequent) constant conditional functional dependencies. *IJDMMM*, 4(3):205–223, 2012.
14. W. Fan. Dependencies revisited for improving data quality. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '08, pages 159–170, New York, NY, USA, 2008. ACM.
15. W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.
16. B. Ganter and S. O. Kuznetsov. Pattern structures and their projections. In H. S. Delugach and G. Stumme, editors, *Conceptual Structures: Broadening the Base, Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, LNCS 2120, pages 129–142. Springer, 2001.
17. B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, Berlin, 1999.
18. G. Graetzer, B. Davey, R. Freese, B. Ganter, M. Greferath, P. Jipsen, H. Priestley, H. Rose, E. Schmidt, S. Schmidt, F. Wehrung, and R. Wille. *General Lattice Theory*. Freeman, San Francisco, CA, 1971.
19. J.-L. Guigues and V. Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Mathématiques et Sciences Humaines*, 95:5–18, 1986.
20. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Computer Journal*, 42(2):100–111, 1999.
21. P. C. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. B)*, pages 1073–1156. MIT Press, Cambridge, MA, USA, 1990.
22. M. Kaytoue, S. O. Kuznetsov, and A. Napoli. Revisiting Numerical Pattern Mining with Formal Concept Analysis. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1342–1347, Barcelona, Spain, 2011. IJCAI/AAAI.
23. M. Kaytoue, S. O. Kuznetsov, A. Napoli, and S. Duplessis. Mining Gene Expression Data with Pattern Structures in Formal Concept Analysis. *Information Science*, 181(10):1989–2001, 2011.
24. S. O. Kuznetsov. Machine learning on the basis of formal concept analysis. *Autom. Remote Control*, 62(10):1543–1564, Oct. 2001.

25. S. O. Kuznetsov. Galois connections in data analysis: Contributions from the soviet era and modern russian research. In B. Ganter, G. Stumme, and R. Wille, editors, *Formal Concept Analysis, Foundations and Applications*, Lecture Notes in Computer Science 3626, pages 196–225. Springer, 2005.
26. S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2–3):189–216, 2002.
27. S. Lopes, J.-M. Petit, and L. Lakhal. Functional and approximate dependency mining: database and fca points of view. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3):93–114, 2002.
28. D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.
29. R. Medina and L. Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In S. Ferré and S. Rudolph, editors, *ICFCA*, volume 5548 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2009.
30. R. Medina and L. Nourine. Conditional functional dependencies: An fca point of view. In L. Kwuida and B. Sertkaya, editors, *ICFCA*, volume 5986 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2010.
31. S. Nedjar, F. Pesci, L. Lakhal, and R. Cicchetti. The agree concept lattice for multidimensional database analysis. In P. Valtchev and R. Jäschke, editors, *ICFCA*, volume 6628 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2011.
32. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. Osborne/McGraw-Hill, Berkeley, CA, USA, 2nd edition, 2000.
33. Y. Sagiv, C. Delobel, D. S. P. Jr., and R. Fagin. An equivalence between relational database dependencies and a fragment of propositional logic. *Journal of the ACM*, 28(3):435–453, 1981.
34. D. A. Simovici, D. Cristofor, and L. Cristofor. Impurity measures in databases. *Acta Inf.*, 38(5):307–324, 2002.
35. D. A. Simovici and R. L. Tenney. *Relational Database Systems*. Academic Press, Inc., Orlando, FL, USA, 1st edition, 1995.
36. S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.*, 36(3):16:1–16:41, Aug. 2011.
37. S. Song and L. Chen. Efficient discovery of similarity constraints for matching dependencies. *Data & Knowledge Engineering*, (0):–, 2013. (in press).
38. S. Song, L. Chen, and P. S. Yu. Comparable dependencies over heterogeneous data. *The VLDB Journal*, 22(2):253–274, Apr. 2013.
39. J. Ullman. *Principles of Database Systems and Knowledge-Based Systems, volumes 1–2*. Computer Science Press, Rockville (MD), USA, 1989.
40. T. Uno, M. Kiyomi, and H. Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In R. J. B. Jr., B. Goethals, and M. J. Zaki, editors, *FIMI*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
41. P. Valtchev, R. Missaoui, and R. Godin. Formal concept analysis for knowledge discovery and data mining: The new challenges. In P. W. Eklund, editor, *ICFCA*, volume 2961 of *Lecture Notes in Computer Science*, pages 352–371. Springer, 2004.
42. R. Wille. Why can concept lattices support knowledge discovery in databases? *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3):81–92, 2002.
43. C. Wyss, C. Giannella, and E. L. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract. In *Proceedings of the Third International Conference on Data Warehousing and Knowledge Discovery*, DaWaK '01, pages 101–110, London, UK, UK, 2001. Springer-Verlag.

Acknowledgements This research work has been partially supported by the Spanish Ministry of Education and Science (project TIN2008-06582-C03-01), EU PASCAL2 Network of Excellence, and by the Generalitat de Catalunya (2009-SGR-980 and 2009-SGR-1428) and AGAUR (grant 2010PIV00057) that allowed professor Napoli to visit the Universitat Politècnica de Catalunya. The second author was partially supported by the MI CNRS

Mastodons program. The authors would like to thank Sergei O. Kuznetsov for his interest and discussions on this work.